Net-Centric Enterprise Services (NCES)

Service Discovery Core Enterprise Services (CES) Architecture

Version 0.4 (Pilot)

March 26, 2004

Prepared for

Defense Information Systems Agency (DISA)

by

Booz Allen Hamilton

This page was intentionally left blank.

Document Change Record

Version Number	Date	Description
0.4	March 26, 2004	First Public Release

This page was intentionally left blank.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

This page was intentionally left blank.

1. EXECUTIVE SUMMARY

The emergence of Web Service technologies has triggered a major paradigm shift in distributed computing: from Distributed Object Architectures (DOAs) to Service Oriented Architectures (SOAs). Within the Department of Defense (DoD) Enterprise there has been a growing need for increased integration and collaboration among "Communities of Interest" (COIs), often across organizational boundaries. The DoD transformation towards Net-Centricity highlights the need even further. A common set of Core Enterprise Services (CESs) represent crucial infrastructure components that support this vision. SOAs are well positioned to become the key technology enabler for Net-Centricity due to their decentralized, loosely coupled, and highly interoperable architecture. To fully take advantage of such SOA benefits, information consumers must have an effective means of discovering the available services (and resources in general), learning about not only their technical definitions, but also their metadata such as capabilities, vocabularies, and policies. Similarly, the service and data providers also need to publish or "advertise" their interfaces and metadata. These publishing and discovery mechanisms must be ubiquitous and dynamic to support the ever-changing mission and business requirements. This document presents a high level architecture to address such needs. The architecture covers two aspects: (1) A reference information model that represents services and service metadata and "profiles" the standard discovery data models (such as those defined in UDDI) for DoD and NCES use; (2) A set of Service Discovery CES that complement industry standards such as UDDI by providing a secure, light-weight, and user-friendly abstraction layer.

Just like the Web Service technologies it leverages, the service discovery architecture presented in this document is still in its infancy. Some potential future work items are listed at the end of the document, and it is expected that the scope of this document will grow over time.

24 *This page was intentionally left blank.*

2. NOTATIONS AND TERMINOLOGY

2.1. Notations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119 [RFC 2119]. E.g.:

> `... they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions) ...`

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

`Fixed width texts used for file names, constants, <XML elements>, and code examples.`

```
Example code listings appear like this.
```

Italics texts are used for variables and other type of entities that can change. Italics are sometimes also used for emphasized text or annotations.

Terms in ***italic bold face*** are intended to have the meaning defined in the glossary.

Underlined texts are used for URLs.

2.2. Terminology

The following terms are frequently used in this document and are briefly explained below using commonly accepted definitions in the security literature. APPENDIX C contains a number of security related glossaries that are much more comprehensive [RFC2828] [WS-GLOS].

Identity: A set of attributes that uniquely identifies a system ***entity*** such as a person, an organization, a service, or a device.

> *Comment: Note that identities are not just for human users. Resources such as data providers and service providers may also have their own identities. Also note that an entity may have multiple identities (e.g. local, legal, organizational).*

Identifier: A sufficiently unambiguous reference to an ***identity*** of a system entity.

> *Comment: Note the difference between an identity and an identifier.*

Principal: As defined in the NCES Security Architecture [SECARCH], a Principal is a system entity that has a ***network identity***, that is capable of making decisions, and to which authenticated actions are done on its behalf. A principal may refer to human entities such as an individual

user, an organization, or a legal entity; depending on the context it may also refer to non-human system entities such as a Web Service provider.

Comment: This document makes a distinction between Principals and Identities. A principal may have multiple local identities in different security domains. For example, a user principal can have a work account called "JDoe" in his employer's network, and also a personal account called "John_Doe" issued by his Internet Service Provider (ISP).

Service Provider: A system entity that provides a collection of services. A service provider usually represents a business or some other organization. A service provider may host one or more Web Services, and may consist of one or many physical machines. If necessary, a service provider may be assigned its own network identity and thus be considered a principal.

Service Consumer: A system entity that issues service requests and consumes returned information. Within a SOA, a service consumer is usually an application.

Comment: A service provider may be a consumer of other service providers.

Service: It is important to note that the term "service" in this document has a broader connotation than ***Web Services*** (with capital letters). We define the latter as services using industry-accepted Web Service technology standards such as SOAP, WSDL, and UDDI, as opposed to general services offered over the Web. Plain ***services***, however, are interpreted in the generic sense and may also represent non-SOAP based resources such as messaging applications, web portals, sensor platforms, or even physical services such as a helpdesk.

Comment: When a service is indeed a Web Service, it is represented by a set of logical interfaces defined using WSDL. Technically, it corresponds to the `<wsdl:Service>` *element in a WSDL document.*

Service Instance: A concrete realization of a service. For Web Services, an instance is sometimes also called an "end point", which denotes a runtime instantiation of a logical Web Service, accessible via a particular technical protocol and transport.

Comment: A service instance is represented in WSDL as a `<wsdl:Port>` *element which references protocol and transport specific service* ***bindings****.*

Discovery Entity: Discovery Entities are data structures that are published for discovery purposes. Discovery entities are expressed in XML and are persistently stored in ***registries*** (defined below). They include data structures that represent service providers, services, service instances, and any discovery related service metadata.

Comment: Discovery Entities, or sometimes referred to as "Entities" in short, are information entities; they are not to be confused with ***System Entities*** *which refer to persons, applications, and devices.*

Registry: A registry is a logical collection of software components (e.g. Web Services) that manages a well-defined set of ***Discovery Entities***.

Comment: A registry is a logical concept; it may physically reside on multiple replicated "nodes" in different systems. From the user's perspective, a registry contains a complete logical copy of the entities, regardless of where those entities are hosted.

Publish: The act of placing one or more entities in a registry by invoking one of the registry's publishing APIs.

*Comment: This document uses the term Publishing and **Registration** interchangeably, although the former is preferred.*

Custody: Each discovery entity in a registry is said to be in the custody of the registry.

Publisher vs. Owner: A ***Publisher*** is the principal who publishes discovery entities in a registry. An ***Owner*** is the principal who has the authority to change a published entity. The publisher is usually the initial owner of the entity, but the ownership may be transferred to another principal.

Comment: In other words, the registry does not own the published entities; it is only the custodian.

108 *This page was intentionally left blank.*

3. BACKGROUND

3.1. Service Oriented Architectures

The emergence of Web Service[1] (WS) technologies has triggered a major paradigm shift in distributed computing. Architectures are quickly moving from DOAs using technologies such as CORBA, DCOM, DCE, and Java RMI, to SOAs using technologies such as SOAP, HTTP and XML. Under a SOA, a set of network-accessible operations and associated resources are abstracted as a "service". The service is described in a standard fashion, published to a service registry, discovered by a service consumer, and invoked by a service consumer. Figure 1 illustrates the three steps of Publish, Discover and Invoke.

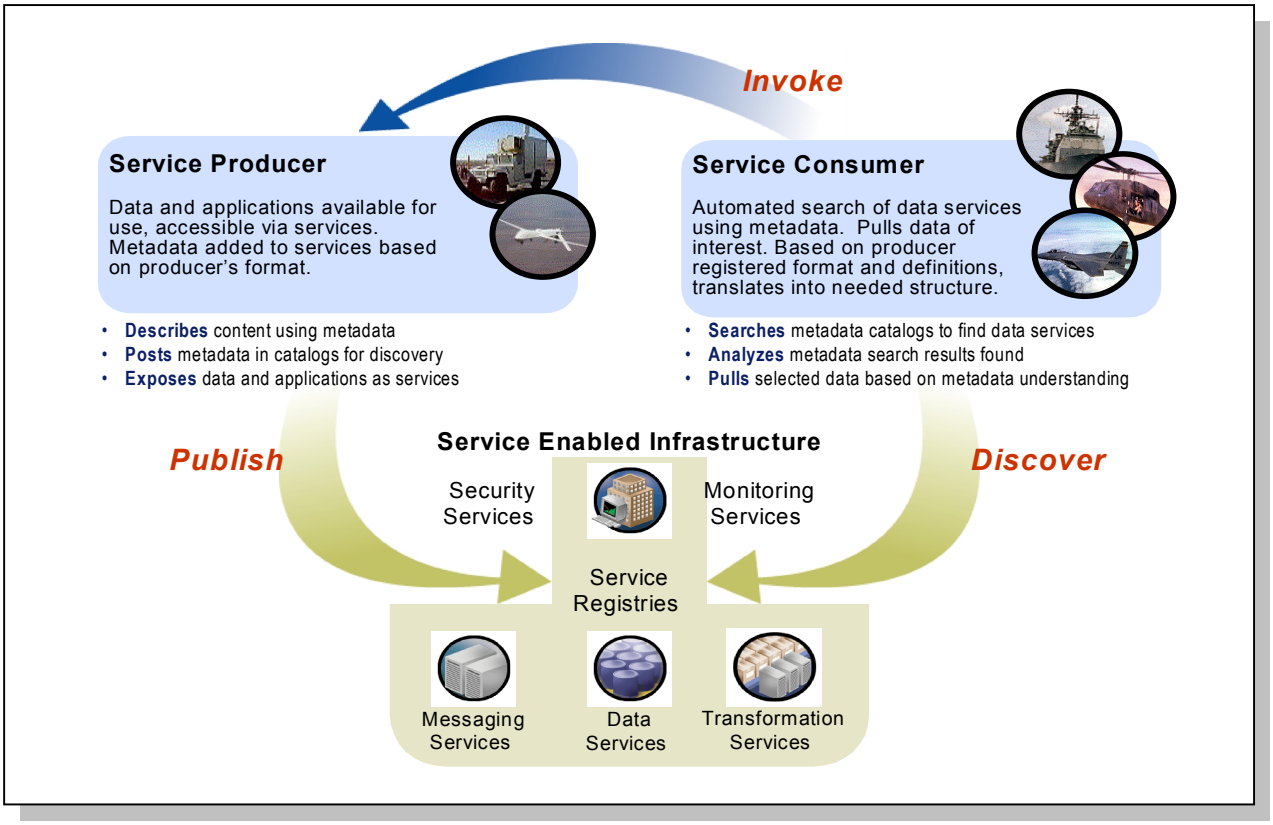


Figure 1: Service Oriented Architecture

Three basic standards serve as the foundation of the Web Services protocol "stack":

- **Simple Object Access Protocol (SOAP)** [SOAP] performs the low-level XML communications necessary for transmitting Web Service calls across the network. SOAP provides a means of XML-based messaging between a service provider and a service consumer.

[1] *This architecture document refers to Web Services (with capital letters) as services using industry-accepted Web Service technology standards such as SOAP, WSDL, and UDDI, as opposed to general services offered over the Web.*

- **Web Service Definition Language (WSDL)** [WSDL] is an XML-based language that defines the functional interfaces for a Web Service. In other words, a WSDL document represents the official "contract" between service providers and their consumers. These WSDL interfaces are described first in abstract message structures, and then bound to a concrete transport protocol and a communication "endpoint".
- **Universal Discovery, Description, and Integration (UDDI)** [UDDI-3] is an emerging standard for organizing and accessing a service registry (see Figure 1). A service registry serves as the yellow pages of a collection of Web Services, providing mechanisms for a service provider to publish its capabilities and for a service user to discover matching services.

A SOA offers several distinct advantages over traditional distributed computing technologies:

- **Maximum Interoperability** – The W3C and OASIS, among others, are currently defining Web Service standards that are entirely based on XML. This ensures that the standards are programming language-, platform-, and programming model-neutral. For example, a .NET Web Service client written in the procedural model of Visual Basic can readily invoke an Object-Oriented Web Service hosted by a Java 2 Enterprise Edition (J2EE) server on a Linux machine.
- **Loose Coupling** – Web Service standards define the functional interfaces that represent the minimal understanding between service consumer and service provider. Knowledge of the service provider is discovered dynamically from a service registry rather than statically coded in the client program.
- **Ubiquity** – Web Service calls are essentially XML messages sent over well-understood Internet protocols such as HTTP. These protocols represent the "least common denominator" of network protocol stacks and makes it easier to overcome firewall and infrastructure constraints. Web Services are likely to be the most viable option for inter-agency information sharing among different autonomous networks.

The migration toward more agile SOAs is not merely a technology push; there are also a number of key business drivers at work. In e-Business and e-Government alike, there is a growing need for increased integration and collaboration across organizational boundaries. Here are some domain scenarios:

- **E-Business / E-Gov Integration** – As businesses strive to keep costs down and become more agile in meeting customer demands, it is necessary to have a technology infrastructure that can enable "deep" integration in the supply chain. Within this scenario, complex systems such as Customer Relationship Management (CRM) and financial systems from manufacturers, suppliers, and distributors can retrieve information and conduct business transactions with one another. For example, a business in the market for a product could shop instantly around the globe for suppliers that meet purchase requirements and dynamically negotiate deals.
- **Counter Terrorism** – There is a pressing need in the intelligence community to provide a highly scalable system that supports collaboration, analytical reasoning and information sharing among multiple Department of Defense, intelligence and federal agencies. Furthermore, obtaining accurate and timely counter-terrorism intelligence requires

processing unprecedented amounts of data, possibly in petabytes, from both classified, unclassified, structured and unstructured sources. There is no single system that can achieve this task and therefore must involve many distributed, decentralized systems.

- **Tactical Warfighting** – Similarly, in the defense sector, there is an increasing need for a C4I (Command, Control, Communications, Computers and Intelligence) system that provides a single, integrated ground picture of forces deployed to the theater. Warfighters need access to real-time information and must operate within the communications infrastructure of existing global networks. Intelligent agents, for example, may automatically discovery and correlate data streams relevant to a current tactical position. DoD's recent Net-Centric Enterprise Services (NCES) initiative reflects this vision.

It is impossible to adopt one single platform, programming language, or protocol that fulfills the needs of these scenarios. A successful architecture must accommodate heterogeneity, and support interoperability in three dimensions: *horizontal* (across peer systems), *vertical* (among different organizational levels) and *temporal* (along a system's evolutionary path). The unique capabilities that come with distributed Service Oriented Architectures can successfully balance these competing dimensions.

3.2. Net-Centric Enterprise Services

Net-Centricity is an architectural mindset that values the relevance, timeliness and accessibility of information above all other qualities. A Net-Centric solution makes data immediately available to those that need it, prohibits unauthorized access to protected resources, and allows consumers to discover relevant information assets without pre-existing knowledge of their existence. The Defense Information Systems Agency (DISA) is currently working to field a set of capabilities that help provide ubiquitous access to reliable, decision-quality information through a net-based Web-Services infrastructure.

There are currently nine Net-Centric Enterprise Services (NCES) defined, and each provides a distinct set of capabilities to the network. Infrastructure services such as Security, Storage, and Enterprise Services Management provide foundational capabilities to other services, while end-user services such as Collaboration facilitate direct communication between people in disparate locations.

With few exceptions, the services defined under NCES are platform- and implementation-agnostic specifications that abstract underlying solutions. The dichotomy formed by splitting the implementation from the specification allows COTS and GOTS implementations to appear and behave the same. That is, given a sufficiently robust specification it's possible to build adaptors to current and future technologies without impacting current integrations. From the system perspective, changes in implementation matter little because they are largely invisible. This model allows for Evolution without convolution.

Moving toward a specification-driven architecture allows for the commoditization of services defined under NCES. Achieving commoditization allows implementations to be tailored to local environments, allows deployments to be more or less robust based on expected load, and ensures that vendors compete on price, reliability and speed, not features. Net-Centricity within NCES values capabilities over implementations, and provides mechanisms that allow each member of

the user community to become a catalyst of change. At the same time, Net-Centric services are reliable, fault-tolerant, secure, and provide unique capabilities that enhance both the structure and substance of the network.

3.3. Overview of Service Discovery Standards

This section provides a brief survey of existing service discovery standards that are applicable in a service-oriented environment. The service discovery architecture described in this document will focus on UDDI for the moment, but its long term goal is to support different discovery specifications and information models in a seamless fashion. The discovery architecture is based on layers of discovery specifications and enabling technologies, as shown in Figure 2 below.

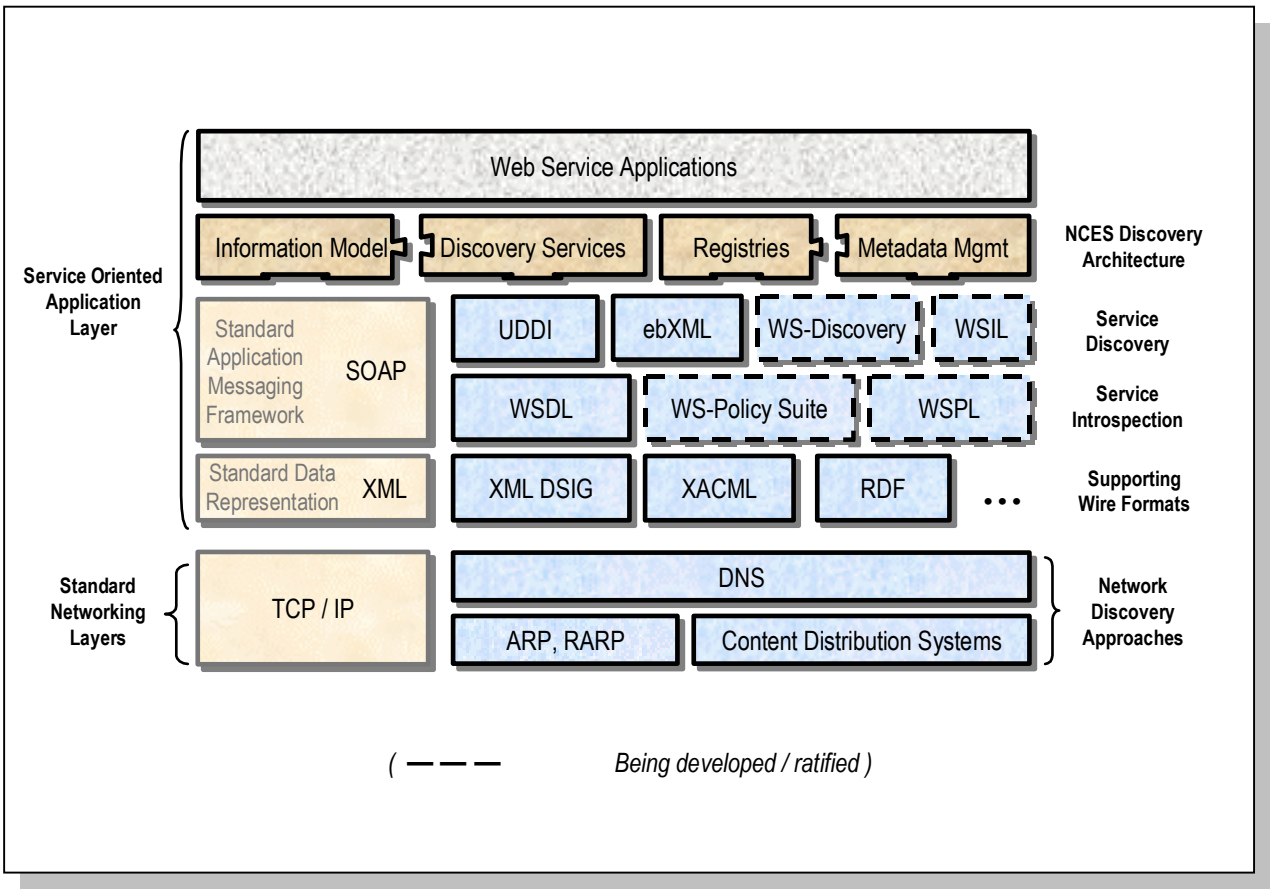


Figure 2: Discovery Technology Stack

3.3.1 UDDI Registries

3.3.1.1 History and Background

The UDDI project began in October 2000 as a collaboration between Microsoft, Ariba, and IBM. Its main goal was to speed interoperability and adoption for Web services through the creation of standards-based specifications for service description and discovery, and the shared operation of a business registry on the Web. Before the UDDI project, there was no industry-wide, accepted approach for businesses to reach their customers and partners with information about their

products and Web services. UDDI enables enterprises to quickly and dynamically discover and invoke Web services, both internally (to the enterprise) and externally.

The initial idea behind UDDI was that software companies, standards bodies, and programmers would populate the public "UDDI Business Registry" with descriptions of different types of services, while businesses would populate the registry with descriptions of the services they support. Marketplaces, search engines, and business applications would then query the registry to discover services at each others' companies. Businesses would also use this data to facilitate easier integration with each other over the Web. UDDI may also be employed as a "private" registry (i.e. behind a firewall) that is hosted by an e-marketplace, a standards body, or a consortium of organizations that participate in a given industry.

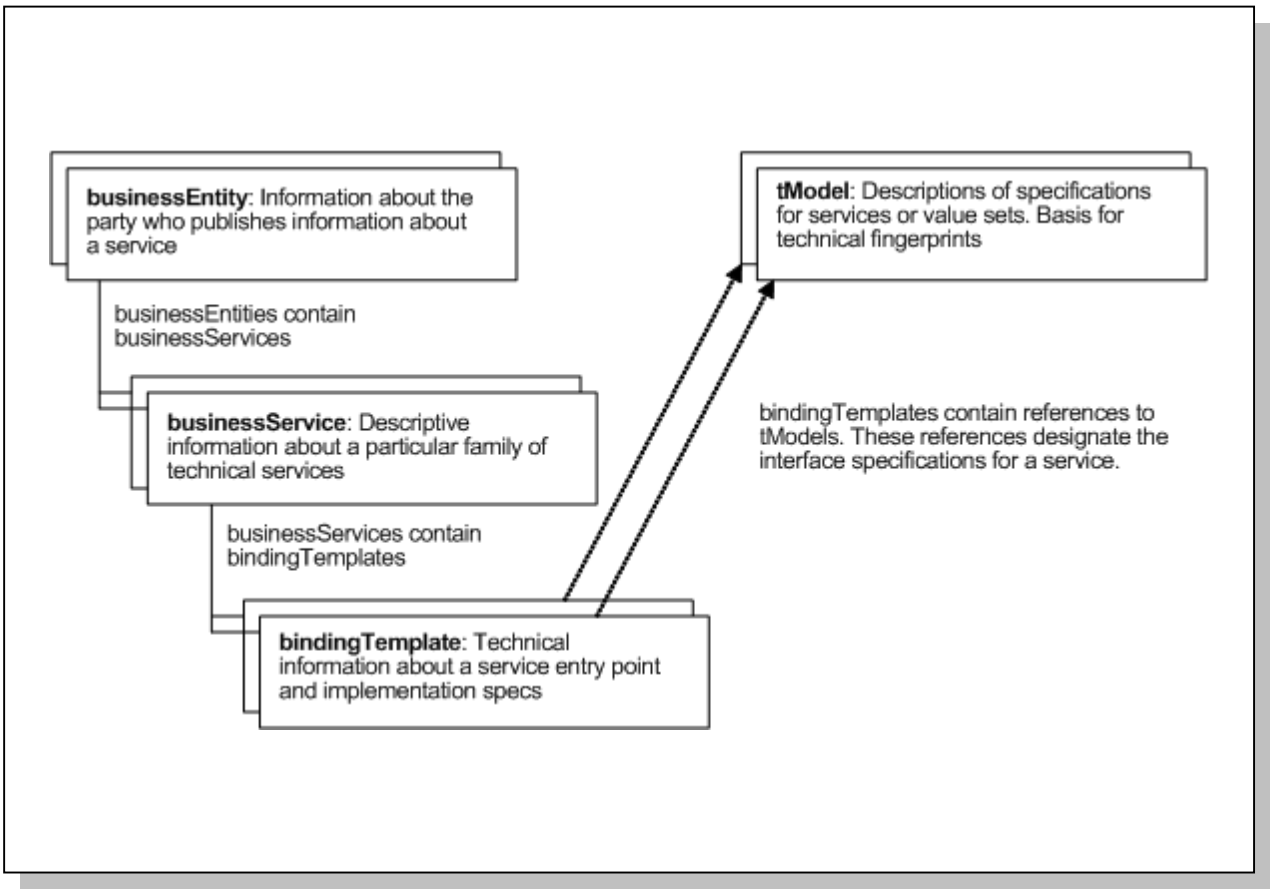
UDDI was moved into the Organization for the Advancement of Structured Information Standards (OASIS) in July 2002. The UDDI Version 2.0 is an OASIS standard, and the Version 3.0 specification (now at v3.0.1, referred to here as v3.0) is a Technical Committee-approved specification as of October 2003.

3.3.1.2 UDDI Information Model

The primary focus of the UDDI information model is business information. The UDDI information model consists of the following four “core” data structures:

- businessEntity
- businessService
- bindingTemplate
- tModel

These structures, and the relationships between them, are represented in Figure 3:



246 **Figure 3: UDDI Information Model**

247 The "core structure" in the UDDI information model is a *tModel*, or "technical model". A
248 tModel represents a reusable concept, such as a Web Service type, a protocol used by Web
249 Services, or a category system. An example of a tModel would be a WSDL document that
250 describes a particular Web Service. Each distinct specification, transport, protocol, or namespace
251 within a UDDI registry is represented by a tModel; this allows tModels to be used to promote
252 interoperability between software systems. It should be noted that a UDDI registry does not
253 actually store the content that is denoted by a tModel, but rather references its location.

254 The *businessEntity* data structure describes a business or other organization that typically
255 provides Web Services. It contains descriptive information about the business or provider and
256 the services it offers, such as:

257 - Names and descriptions in multiple languages
258 - Contact information
259 - Classification information

260 The *businessService* data structure represents a logical grouping of Web Services that a business
261 provides. It should be noted that at this level, there is no technical information provided about
262 these services - rather, this structure allows the ability to assemble a set of services under a

common rubric. An example of a businessService would be a set of Purchase Order Web Services (submission, confirmation, and notification) that are provided by a business.

The structure that is used to describe the technical information about a Web Service is known as a *bindingTemplate*. Each bindingTemplate structure represents an individual Web Service, and contains either the access point for a given service, or an indirection mechanism that will lead one to the access point.

3.3.1.3 UDDI V3.0 Features

The UDDI Version 3.0 specification contains features that render it quite different from the UDDI Version 2.0 specification. Some of these features are:

- Multi-Registry Support: Previous versions of UDDI did not permit the publishing of across multiple registries. This capability is now possible with Version 3.0.
- Digital Signature Support: Allows UDDI entities to be digitally signed, thereby contributing a higher level of data integrity and authenticity
- Policies: Enables various decisions to be enforced by policies, contributing to more consistent handling of contents
- Publish/Subscribe: A new Subscription API includes robust support for synchronous or asynchronous notification of registry events to users

3.3.2 ebXML Registries

3.3.2.1 History and Background

The ebXML Registry specification was created as part of the 18-month ebXML initiative that ended in May 2001. Sponsored by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) and OASIS, ebXML is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. ebXML provides companies with a standard method to exchange business messages, conduct trading relationships, communicate data in common terms, and define and register business processes. An ebXML registry provides a mechanism by which XML artifacts can be stored, maintained, and automatically discovered, thereby increasing efficiency in XML-related development efforts.

The OASIS/ebXML Registry Technical Committee was created in May 2001 to build on the ebXML initiative efforts. The ebXML Registry Version 2.0 specification is an OASIS-approved specification, and the ebXML Registry Version 3.0 specification is in the final phases of development. The ebXML Registry specification is actually comprised of two specifications - ebXML Registry Information Model (ebRIM) and ebXML Registry Services (ebRS). We refer to these specifications collectively here as the "ebXML Registry specification".

3.3.2.2 ebXML Registry Information Model

Unlike UDDI whose primary focus is business information, the main focus of the ebXML Registry information model is more general to encompass XML and non-XML artifacts. Therefore, the ebXML Registry information model is more abstract in nature than that of UDDI.

The ebXML Registry information model consists of two "core" data structures, or *classes*:

- RegistryObject
- RegistryEntry

A *RegistryObject* provides metadata for a stored *RepositoryItem* (the term used to refer to that actual object that is stored) – such as name, object type, identifier, description, etc. A RegistryObject can represent many different types of RepositoryItems, from XML schemas, to classification schemes, to Web Service definitions.

In contrast, a *RegistryEntry* is used to represent "catalog-type" metadata about RepositoryItems – that is, metadata about the current state of a RepositoryItem in the registry (e.g. version, status, stability). Consequently, the metadata associated with a RegistryEntry is (in general) more "fluid" than that associated with a RegistryObject. The RegistryEntry class inherits from the RegistryObject class.

3.3.2.3 ebXML Registry Within ebXML Technical Architecture

The ebXML Registry is a central component of the ebXML Technical Architecture, as it serves as a storage facility and discovery mechanism for the various artifacts that are necessary for engaging in electronic business using the ebXML framework. This is shown in Figure 4:

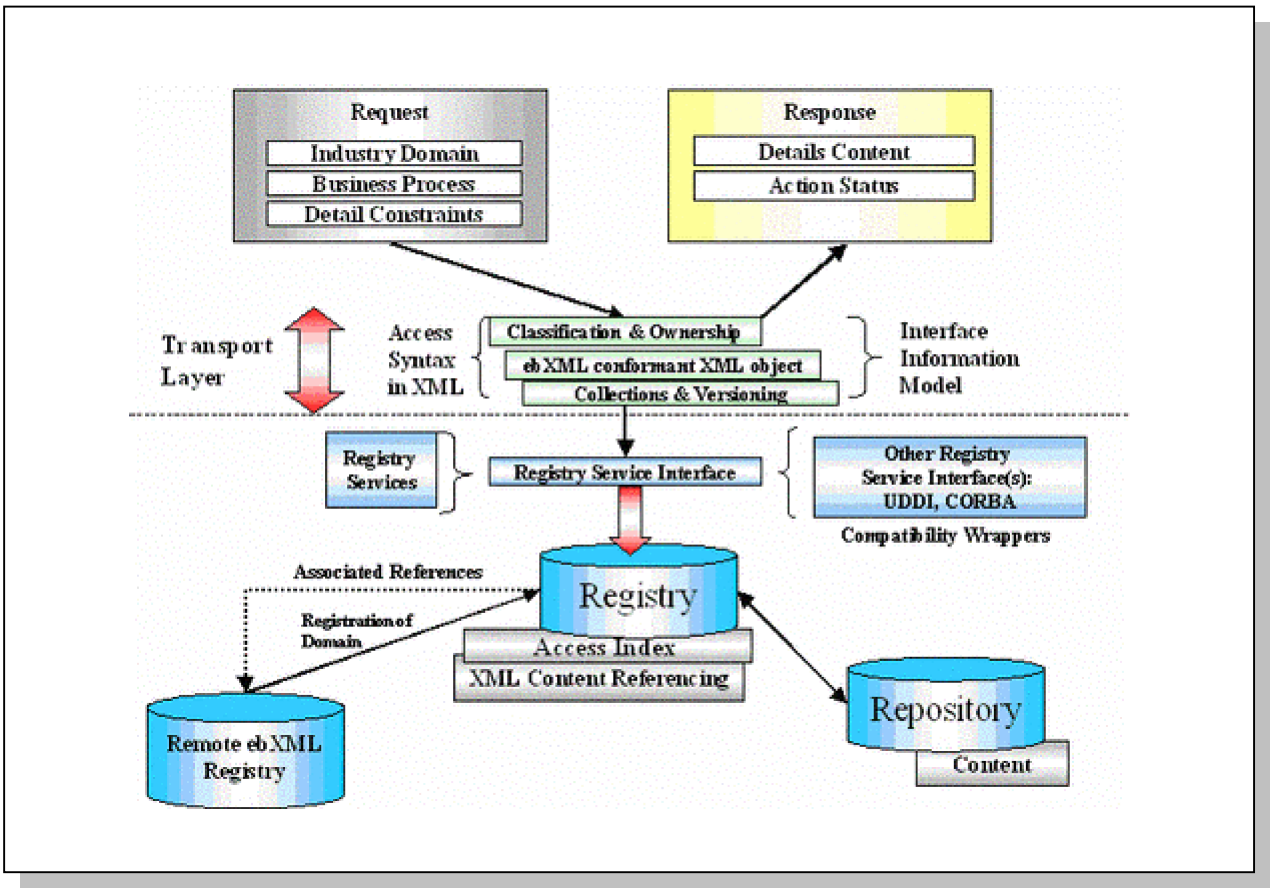


Figure 4: ebXML Technical Architecture

In the figure above, an ebXML registry interacts with both a local repository and a remote ebXML registry. Requests are sent to the registry and responses are received from the registry through a Registry Service Interface. The Registry Service Interface may interact with other Registry Service Interfaces, such as UDDI, and open interface standards such as Common Object Request Broker Architecture (CORBA).

3.3.3 Enabling Technologies

To ensure that service discovery is protected by proper authentication and authorization mechanisms under enterprise security policies, the following standards are also relevant. Please refer to the NCES Security CES Architecture document [SECARCH] for detailed discussions on these standards.

- **WS-Security**, short for Web Services Security, is a standard jointly proposed by an industry consortium (IBM, Microsoft, and Verisign) and currently being ratified by OASIS [WSS]. It serves as the foundation to address SOAP-level security issues, with three major propositions: (1) use of security tokens in SOAP headers for user identity and authentication, (2) use of XML-Signature standard for message integrity and authenticity, and (3) use of XML-Encryption for message confidentiality. There are of course many other security requirements that are not yet addressed by WS-Security. WS-Security is only the first in a series of standards proposed by the consortium aimed at providing a

broader security framework for Web Services. Additional standards and vendor proposals are forthcoming that address issues such as authorization, privacy, policy, trust, secure conversation, and federation.

- **XML-Signature**. A formal Recommendation (i.e. approved standard) from W3C [XMLDSIG], this spec covers the syntax and processing of digitally signing selected elements in an XML document using either symmetric (secret) key or asymmetric (public/private) key cryptography. Such digital signatures help ensure the data integrity of the signed XML elements so that any unauthorized data modifications can be detected via signature verification.
- **XML Access Control Markup Language (XACML)**. Ratified as an OASIS standard in February 2003 (1.0 version), XACML defines a generic authorization architecture and the constructs for expressing and exchanging access control policy information using XML. Policy constructs include policies, rules, combining algorithms, etc. Like SAML, XACML also provides a request/response semantics for authorization decisions to facilitate access control mechanisms.

Table 1 lists the versions of the specifications supported in this architecture document:

Table 1: Currently Supported Standards and Their Versions

Specification	Version
SOAP	1.1
WSDL	1.1
UDDI	2.0, with partial 3.0 features such as subscription
UDDI Tech Note on Taxonomy	2.0, 2001-07-17
UDDI Tech Note on WSDL Mapping	2.0, 2003-11-03
WS-Interoperability	Basic Profile 1.0a
XACML	1.1
XML-DSIG	W3C Recommendation 2002-02-12

4. ARCHITECTURE OVERVIEW

4.1. The Need for Enterprise Service Discovery

In a Net-Centric environment, service discovery plays a critical role:

- It allows information **producers** to publish / advertise resource definitions, descriptions, metadata, and accessibility. The information producers may include Web Services (e.g., service-enabled target tracking applications), data repositories (e.g., a Coalition Shared Database), devices (e.g., sensor platforms), or even non-technical business functions (e.g., a helpdesk for technical support).
- It allows information **consumers** to discover, retrieve, and manage service information as advertised by producers. The information consumers may include thick clients (e.g., service-enabled Command and Control applications), thin clients (e.g., web browsers), or devices (e.g. PDAs).
- It should allow information **developers** to transparently enhance discovery, retrieval, and publishing services without interrupting normal business operations.

Along with other NCES discovery CESs such as Person Discovery and Content Discovery, Service Discovery is responsible for **getting the right information to the right people at the right time in the Net-Centric environment**.

A service "Yellow Page" is often the analogy used to describe service discovery. This analogy, however, is not sufficient to convey the following important characteristics of Enterprise Service Discovery in a dynamic SOA setting:

- Service Discovery makes use of **common registries** to facilitate information awareness, access, and delivery. Unlike yellow pages that are easily duplicated and distributed to consumers, the registries are logically centralized repositories (though they may be physically replicated to a certain degree), managed by the enterprise.
- Unlike yellow pages that are infrequently updated and relatively static, the service registries contain **dynamically updated** information.
- Service Discovery is not merely about service definition and location. It relies heavily on a variety of **service metadata**. Such metadata plays a crucial role for service visibility and accessibility (more on this later).
- Service Discovery is not just for end users browsing and searching services. It is also (if not more) geared towards dynamic, **runtime discovery** by applications and other system entities.
- To support a Net-Centric environment with heterogeneous platforms, products, protocols, and programming languages, it is a given that Service Discovery should be based on **open standards** to ensure interoperability.
- Service Discovery should be properly **secured** so that sensitive information in the registries is protected and that the publishing and discovery of such information are subject to proper access control.

4.2. Summary of Architectural Requirements

The primary goal of the service discovery architecture defined in this document is to support dynamic publishing, discovery, and introspection of enterprise services as well as their metadata. The high level requirements generally fall into the following areas of service publishing, service inquiry, service metadata management, and security integration. Many of the requirements are going to be addressed in this version of the architecture, but some are long term goals that will be addressed in the near future.

4.2.1 Service Publishing

Service publishing involves placing in a registry such discovery entities as service providers, services, service instances, as well as the technical "fingerprints" of the service instance which represent all relevant metadata. The following mechanisms need to be supported:

1. **Manual service publishing** – A human user / operator serves as the publisher, who uses a web user interface to publish the service entities in the registry. This is the most straightforward approach and is likely to be the primary way of publishing services in the near term.
2. **Automated service publishing** – In this case, the publisher is an application (possibly the service itself), which uses a publishing Web Service / API provided by the registry to publish the service entities. The registry can obtain most of the service information by introspection of its WSDL definitions, as described later in this document.
3. **Dynamic updates to discovery entities** – In addition to automated publishing, a service may need to dynamically update its definitions and metadata in the registry, so that the entities in the registry is kept in sync with the operating conditions of the real service. For example, when a service instance is moved to a new end point, or a new authentication requirement is instigated, the service entry in the registry needs to be updated accordingly.

4.2.2 Service Inquiry

1. **Manual, user-oriented service inquiries** – Individual users and developers need to be able to browse, search, and inspect services and other entities via a web based user interface.
2. **Dynamic, runtime service inquiries** – Service consumers may also need to discover services at runtime, using an inquiry Web Service interface provided by the registry. Dynamic inquires are crucial to achieve *location transparency* of services, which allows service consumers to connect to a provider even if the service's location (or even transport) is changed.
3. **"Persistent" service inquires** – In some cases a service consumer may want to be kept up to date on certain discovery entities published in a registry. A consumer should be able to advertising such a need by subscribing to changes of the interested discovery entities, and get notified of such changes near real time.

Regardless of the inquiry mechanisms, the service inquiries should have enough expressive power to support not just name and identifier based queries, but also complex query constraints based on arbitrary metadata.

4.2.3 Service Metadata Management

As mentioned earlier, Enterprise Service Discovery will heavily leverage metadata about service providers, services, and service instances. For both publishing and inquiries, the following levels of metadata need to be supported:

1. **"White Page"** metadata, which are basic *resource definitions* such as identifiers, names, locations, and interface definitions.
2. **"Yellow Page"** metadata, which are *content related metadata* describing what the service provides. Such metadata attributes may include subjects, key words, service types, categories, temporal and spatial constraints, or other "Dublin Core"-like attributes.
3. **"Brown Page"** metadata, which are *"semantic" level metadata* describing business / functional capabilities provided by the service. Such metadata may refer to well-defined business vocabularies, taxonomies, ontologies, XML schemas, domain data models, or business rules. This kind of metadata is especially important in specialized Communities of Interest (COIs) where community members often need to interact in an ad-hoc fashion but may not necessarily talk the same "business language". Publishing such metadata helps information providers and consumers understand one another's capabilities and also enables third-parties to mediate among them as necessary, achieving free information flow that wouldn't be possible among stove-piped systems.
4. **"Green Page"** metadata, which describes the *access capabilities* of a service, that is, technological and environment related metadata required for accessing the service. Such metadata may include, but not limited to, security or "Quality of Protection" (QoP) requirements, Quality of Service (QoS) attributes, transport protocol details, and so on. For example, an imagery service may want to advertise that consumers must have T1 and above bandwidth to access the service, or that access to downloading certain imagery files will only be granted during non-business hours.

All the above metadata may be expressed in XML.

In order to support such metadata requirements, registries must have the following capabilities:

1. **Management of business and technical vocabularies and taxonomies**, via both Graphic User Interfaces and programmatic APIs. UDDI, for instance, provides strong tModel and taxonomy management features that may be used to manage most if not all types of the above metadata.
2. **Seamless integration with existing XML-based metadata repositories**. Service metadata attributes may already exist elsewhere in the system. For instance, there may exist certain Resource Attribute Services, defined in the NCES Security Architecture [SECARCH], which may serve for discovery purposes. Eventually the service registries also need to integrate with other DoD metadata initiatives such as the Defense Discovery

Metadata Specification (DDMS) and the DoD XML Repository, so that existing XML schemas and taxonomies may be reused for service discovery purposes.

3. **Metadata-driven service publishing and inquiry**. This is essential because regardless of whether static (design time) or dynamic (runtime) approaches are used, the effectiveness and accuracy of service inquiries will depend largely on the richness of metadata.

4. **Semantic metadata integration**. In the long run, as the number of taxonomies increases, the consistency and interoperability among them may become critical for effective and intelligent service discovery. Semantic Web technologies such as Web Ontology Language (OWL) may be used to build semantic "bridges" among disparate business taxonomies. Future registries may need to support semantic representations of service definitions and metadata.

4.2.4 Security Integration

Service discovery is closely related to service security, and the relationship goes two ways: On one hand, service publishing and inquiry APIs as well as the registries themselves need to be protected by the enterprise security mechanisms and policies. Conversely, establishing the proper trust relationship between a service consumer and a provider often involves discovery of each other's security characteristics. The following requirements cover both these aspects:

1. **Securing the Discovery Service Interfaces** – For both publishing and inquiry, the service interfaces need to be protected using the techniques prescribed in the NCES Security Architecture [SECARCH], so that:
   - Identities of publishers, inquirers, and discovery service providers may be established;
   - The publishing and inquiry requests and responses are authenticated and their message integrity verified;
   - The requests and responses are authorized against access control policies, if necessary.
2. **Trustworthiness of the Discovery Entities** – As mentioned earlier, a service registry is the custodian of the discovery entities places within it, and the owner (usually the publisher) who is "advertising" the service is ultimately responsible for the quality, timeliness, and authenticity of these entities. The owner needs to be able to "vouch for" its published entities so that consumers can have some degree of trust on these entities. Digital signatures, for example, may be applied to certain entities to achieve this end. In addition, the registry may wish to maintain consumer feedback and ratings on services, which may be especially helpful in ad-hoc, COI-oriented settings.
3. **Protection of Sensitive Discovery Entities** – When rich sets of service metadata are captured in a service registry, extra care must be taken to make sure sensitive information is not released to unauthorized parties. The registry must integrate with the Security CES Policy Services which manage authorization policies for the enterprise.
4. **Discovery of the QoP requirements of services** – As already outlined in Section 4.2.3.

4.2.5 Ubiquitous Service Discovery

Service discovery in the Net-Centric environment involves much more than setting up a service registry. For one thing, a centralized service registry cannot sufficiently scale, both functionally and topologically, to support decentralized agencies and user communities with different discovery requirements and different IT infrastructures (more on this later). Furthermore, the loosely coupled nature of NCES requires a ubiquitous, fast, "always-on" kind of discovery capability. The following goals are necessary to support this vision:

1. **Decentralized Service Registries** – Local service discovery activities need not go through WAN connections to a centralized registry.
2. **Discovery across Multiple Registries** – The architectural approaches will be further explored in the next version of this document.
3. **Discovery Bootstrapping** – When a service consumer "plugs in" to a COI network, it first needs to locate the service registry before it's able to further discover other service providers. In the near term, the consumer may need to be configured a priori with the registry location. In the long term, however, automatic bootstrapping mechanisms (e.g., the multi-cast probe messages proposed by WS-Discovery) may be employed to provide ubiquitous discovery in an ad-hoc network.

4.3. Scope, Assumptions, and Limitations

The following assumptions and limitations have been identified for the NCES Service Discovery 0.4 Release:

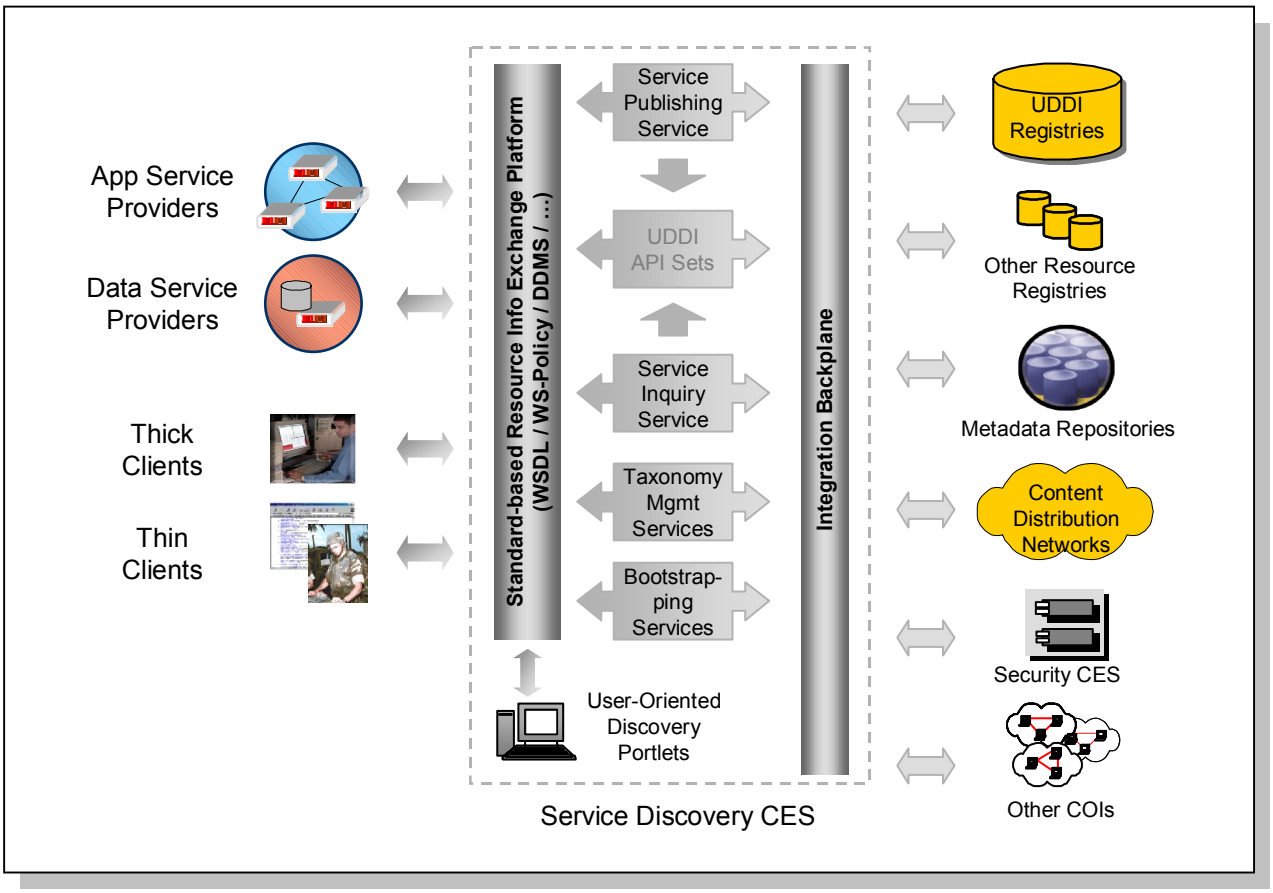
1. The architecture does NOT mandate or prescribe the **data replication** mechanisms among the nodes in a registry. A UDDI registry, for example, may choose to use the SOAP-based replication APIs or backend database replication scripts.
2. The document does NOT yet cover the federation / affiliation of **multiple registries** (e.g. registries from different trust domains).
3. The document does NOT yet address the **integration with other resource registries** such as the DoD XML Repository, ebXML registries, or other non-Web Service registries.
4. The document does NOT yet support the Defense Discovery Metadata Specification (**DDMS**).
5. The architecture currently does not define **registry subscription and notification interfaces**. This will be addressed in the next version.
6. The architecture currently does not define SOAP-based **taxonomy management interfaces**. This will be addressed in the next version.
7. The architecture does NOT yet address the automatic registry **bootstrapping** issue. Bootstrapping to a service registry in a trust domain may rely on either a well-known network address for the registry and / or pre-configured registry location(s) in service consumer applications.

8. The information model defined in this document does not yet address the **versioning of services**. Different services versions are currently treated as different services.
9. The information model defined in this document does not yet support **mapping WSDL *operations* to UDDI**. This will be addressed in later versions.
10. This architecture does NOT yet support **mapping WSDL *extensions*** (such as WS-PolicyAttachment) **to UDDI**. This will be addressed in later versions.
11. This architecture does NOT yet support **digitally signed entities** by owner. This will be addressed in the near future.
12. This architecture does NOT yet support **semantic service discovery**.

4.4. Conceptual Service Discovery Architecture

Figure 5 presents a very high level illustration of the service discovery architecture. The diagram reflects the following concepts:

1. Service consumers and providers (shown on the left side of the diagram) exchange discovery related information (e.g. service descriptions) with the Service Discovery CES through open industry standards such as WSDL and UDDI, as well as through potential DoD-wide information standards such as DDMS.
2. Although the architecture exposes straight UDDI APIs to service publishers and inquirers, other "value-added" discovery services are defined to provide streamlined service publishing, business user-friendly inquiries, and advanced features such as DoD-specific taxonomy management. In fact, these services are the RECOMMENDED discovery interfaces over straight UDDI APIs, for reasons outlined in the next subsection. These component interfaces together constitute a platform-independent abstraction layer called ***Service Discovery CES*** (shown in the middle section of the diagram).
3. The discovery services leverage existing NCES infrastructure (shown on the right side of the diagram) such as metadata repositories and the Security CES through an integration backplane.



572 **Figure 5: Conceptual Enterprise Service Discovery Architecture**

573 The Service Discovery CES offers the following components:

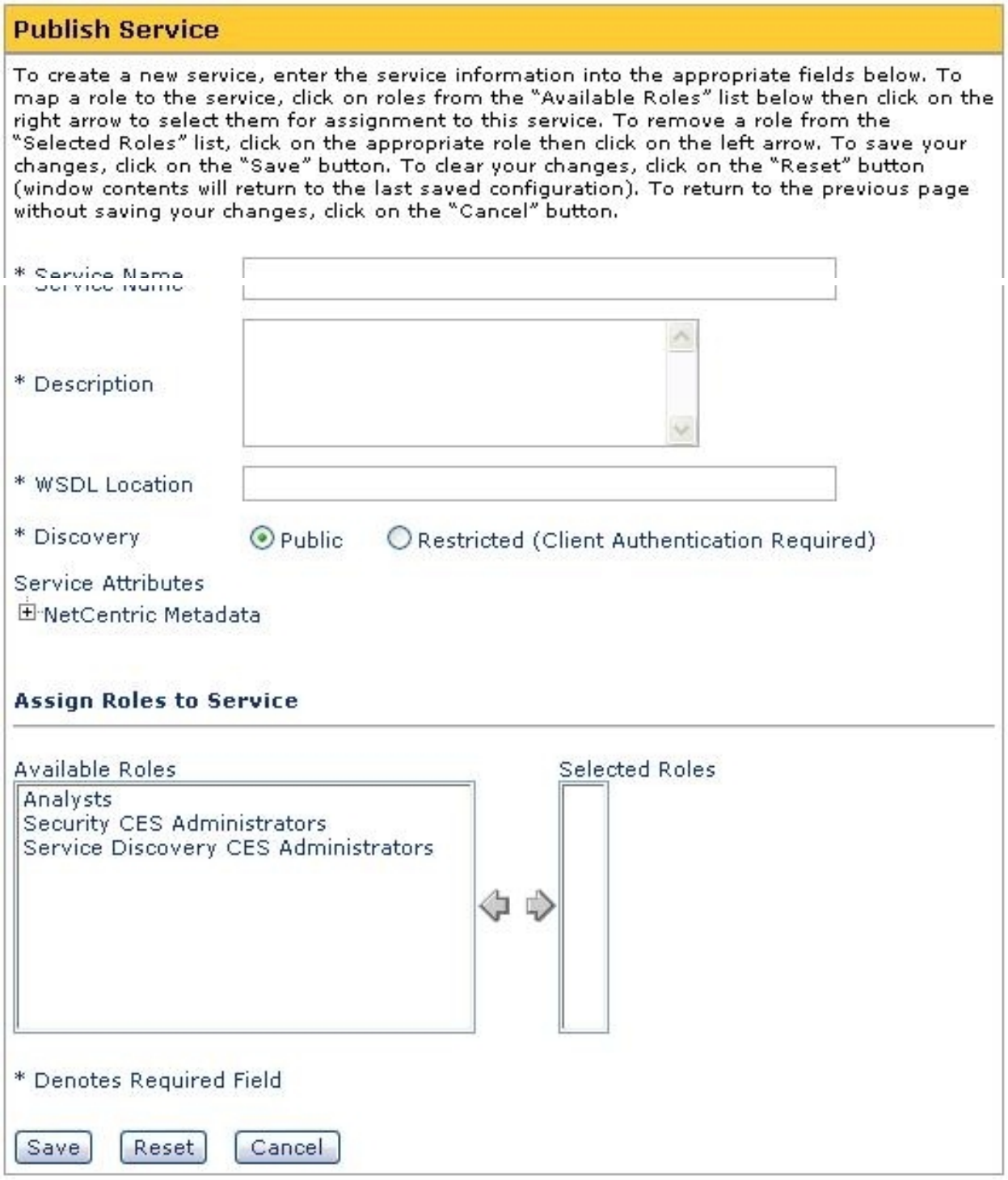
574 ➢ **Service Publishing Service** (SPS). This Web Service provides operations for
575 publishing, un-publishing, and updating service related entities in the registry. It serves
576 as a "one-stop shop" for service publishers, bundling processing steps from parsing the
577 WSDL description to creating tModels to categorizing the service with required
578 taxonomies. A sample XML listing for publishing a service is included in Appendix B.2.

579 ➢ **Service Inquiry Service** (SIS). This service provides simple yet powerful search
580 interfaces for services, using simple XML constructs such as a "serviceLocator". It
581 shields inquirers from the highly technical UDDI terminology and query semantics (e.g.,
582 tModel keys, category bags, bindingTemplates, and so on), and in some cases also
583 optimizes the search by combining what would otherwise be several UDDI calls into a
584 single inquiry. APPENDIX B contains a number of examples of using the Inquiry
585 Service.

586 ➢ Other advanced service interfaces such as those for taxonomy management and
587 bootstrapping will be defined in the future.

588 ➢ **A set of user-oriented service discovery portlets** are also provided to assist business
589 users or developers who are not familiar with UDDI technology to perform design-time

590 service publishing and inquiry tasks. Figure 6 illustrates a sample service publishing
591 screen shot.



592 **Figure 6: Sample Portlet Screen**

The SPS and SIS will be defined in details in a separate *Service Discovery CES Specification* document.

The discovery architecture provides true loose coupling among applications and enhance overall system stability. As can be seen from Figure 5, this is reflected in the "plug and play" capability for both discovery consumers and discovery infrastructure providers: On the left hand side of the architecture diagram, service providers and consumers can easily plug in to the discovery framework because all interfaces are fully standards based. The right hand side of the diagram illustrates how developers can swap registry implementations without affecting the Web Services and end users. This is made possible because the discovery service interfaces in the middle remain the same.

To reiterate, the service interfaces defined by this architecture are *specifications*, not implementations. The actual implementations may utilize best-of-breed COTS and GOTS technologies and may vary in different IT environments, but the specifications will remain stable and interoperable. Going forward it is envisioned that the specifications will be driven by the collective efforts of various NCES initiatives and their requirements, while at the same time reflecting current industry best practices.

4.5. Why Discovery Services?

There are many reasons why a service discovery abstraction layer is preferred over using the native UDDI APIs. This subsection attempts to explain some of them from several different perspectives:

4.5.1 The Interoperability Perspective

Basing the NCES service discovery on the UDDI specification alone may create interoperability problems with other existing or potential discovery standards:

- As outlined in Section 3.3, ebXML is also a widely recognized industry standard for discovery;
- New standards such as WS-Discovery [WS-DISC] are being proposed and may become relevant to NCES;
- For discovery of non-Web Service resources, it may become necessary to integrate / "wrap" other discovery mechanisms such as Java Naming and Directory Interface (JNDI), CORBA Naming Service, JINI, and Peer-to-Peer (P2P) discovery protocols. This cannot be easily achieved with UDDI interfaces.

4.5.2 The Usability Perspective

To be able to effectively publish and search the UDDI registry, one has to have thorough knowledge on the UDDI information model, which is quite comprehensive yet in some places not very intuitive. Take the tModel concept for example, it is relatively easy to grasp its usage as a "label" for representing a technology "fingerprint", such as compliance to a certain specification, but its general use as a type system for taxonomies and many other internal UDDI constructs are not very well understood. As a result, service discovery tasks, especially dynamic

runtime inquiries, are not always easy to perform from the consumer's perspective. The following are just a few examples:

- Composing UDDI queries in general is not an intuitive task. The use of findQualifiers, categoryBags, keyedReferences, as well as somewhat cryptic tModel keys. See Appendix B.1 for an example;
- In some cases, a business inquiry has to be implemented as several sequential UDDI queries, which is tedious and may cause performance concerns. For instance, an inquiry such as "find all SOAP implementations of a particular service spec" would result in at least three UDDI queries, as outlined in a UDDI Technical Committee's Technical Note on using WSDL in a UDDI registry [UDDI-WSDL, Section 3.3.5].
- In general, the "drill-down" style of UDDI query APIs may be good for design time, interactive UI-based queries, but may not be suitable for runtime queries. For instance, many "find all" type of inquiries may result in N+1 UDDI calls.

Similar conclusions may be drawn for publishing interfaces.

There are also other technologies that attempts to simplify the UDDI APIs, such as the Java API for XML Registries or JAXR [JAXR], however it is a programming language-specific abstraction layer that cannot be used by other frameworks such Microsoft .NET.

4.5.3 The Information Integrity Perspective

As mentioned, the UDDI information model is very powerful and flexible enough to support a wide variety of business needs. The specification itself only enforces a minimal number of data elements, leaving most things optional and up to the business to enforce. For the DoD enterprise, it is then imperative to "profile" the usage of the UDDI spec according to DoD and Net-Centricity needs, so that all necessary information is captured and properly formatted to meet quality and interoperability requirements.

More specifically, for service publishing, at least the following processing steps are involved:

- Creating a businessEntity for the service provider, if it doesn't exist;
- Creating tModels for service portType(s) as well as operations within the portTypes;
- Creating tModels for service bindings;
- Creating a businessService construct for the service;
- Tagging the service with required categorization attributes based on NCES taxonomies (defined in Appendix A.1);
- Creating bindingTemplate constructs for service instances;
- Tagging bindingTemplates with required technical "fingerprints" (e.g. security policies) based on NCES taxonomies (defined in Appendix A.2);
- Creating the necessary access control policies in the Security CES.

These steps need to be performed using a series of UDDI calls in the correct order, following the processing rules defined in Section 6. Furthermore, the referential relationships among these data elements need to be established and carefully maintained.

The burden of enforcing such data integrity requirements will fall unto the service publishers unless a higher level service can streamline this process.

4.5.4 The Security Perspective

Currently commercial UDDI products all provide their own proprietary security features, which cannot be seamlessly integrated with the Security CES without significant customization efforts.

By contrast, the SPS, SIS, and other discovery CES components can easily integrate with the security CES through, for example, the deployment of security handlers. Integration with the Security CES will be discussed in Section 7.

UDDI 3.0 does provide an open security architecture for employing external Policy Decision Points (PDP) and Policy Administration Points (PAP). Until UDDI 3.0 becomes an official standard and widely implemented in vendor products, however, the discovery CES layer remains the only viable approach for providing enterprise security for service registries.

4.5.5 The System Architecture Perspective

Last but not the least, the need for ubiquitous discovery presence as outlined in Section 4.2.5 calls for a light-weight discovery CES layer that has the following advantages:

- Replication of UDDI registries is easy to implement but may prove costly, because the physical environment for deploying these *enterprise* registries has to meet specific requirements on capacity, performance, availability, bandwidth, as well as security. By contrast, the discovery services are applications with small footprints and can be distributed much more easily to many network domains.
- The discovery CES may serve as the discovery "proxy" in the local network so that local consumers need not make more costly discovery requests across the network boundary (or even WANs). The discovery CES may also be able to cache inquiry results to improve performance.
- When multiple federated registries are involved, the local discovery CES may transparently redirect inquiries to affiliated remote registries. From the consumer's perspective, it is just a single "virtual" registry.

In short, the discovery CES layer is more "agile" and complements enterprise-strength UDDI registries. Together they form a discovery framework that is highly scalable and resilient.

This page was intentionally left blank.

5. SERVICE DISCOVERY USAGE SCENARIOS

A typical usage scenario for Discovery Services is a *publish-find-bind* cycle. At a high-level, the scenario is described as follows:

1. A service provider publishes a service as well as its deployed instances to the Service Discovery CES.
2. A service consumer searches through Service Discovery CES and finds the service instance(s) that meet the search criteria.
3. The service consumer uses the end point information of a found service instance to "bind to" and consume the service.

Figure 7 illustrates the publish-find-bind usage scenario.

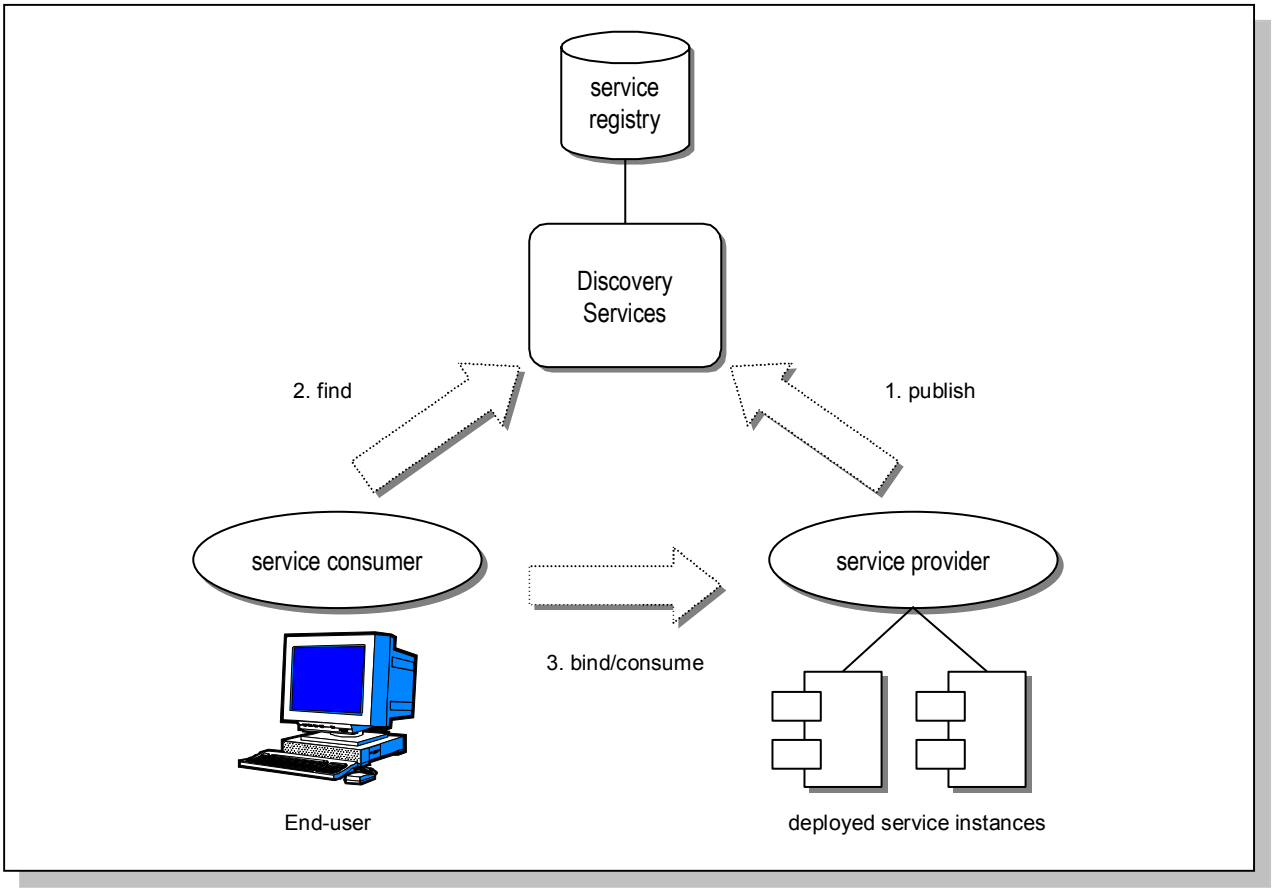


Figure 7: Typical Service Discovery Cycle

5.1. Static Service Discovery

Figure 8 describes a common usage scenario that involves developers make service inquiries at design time.

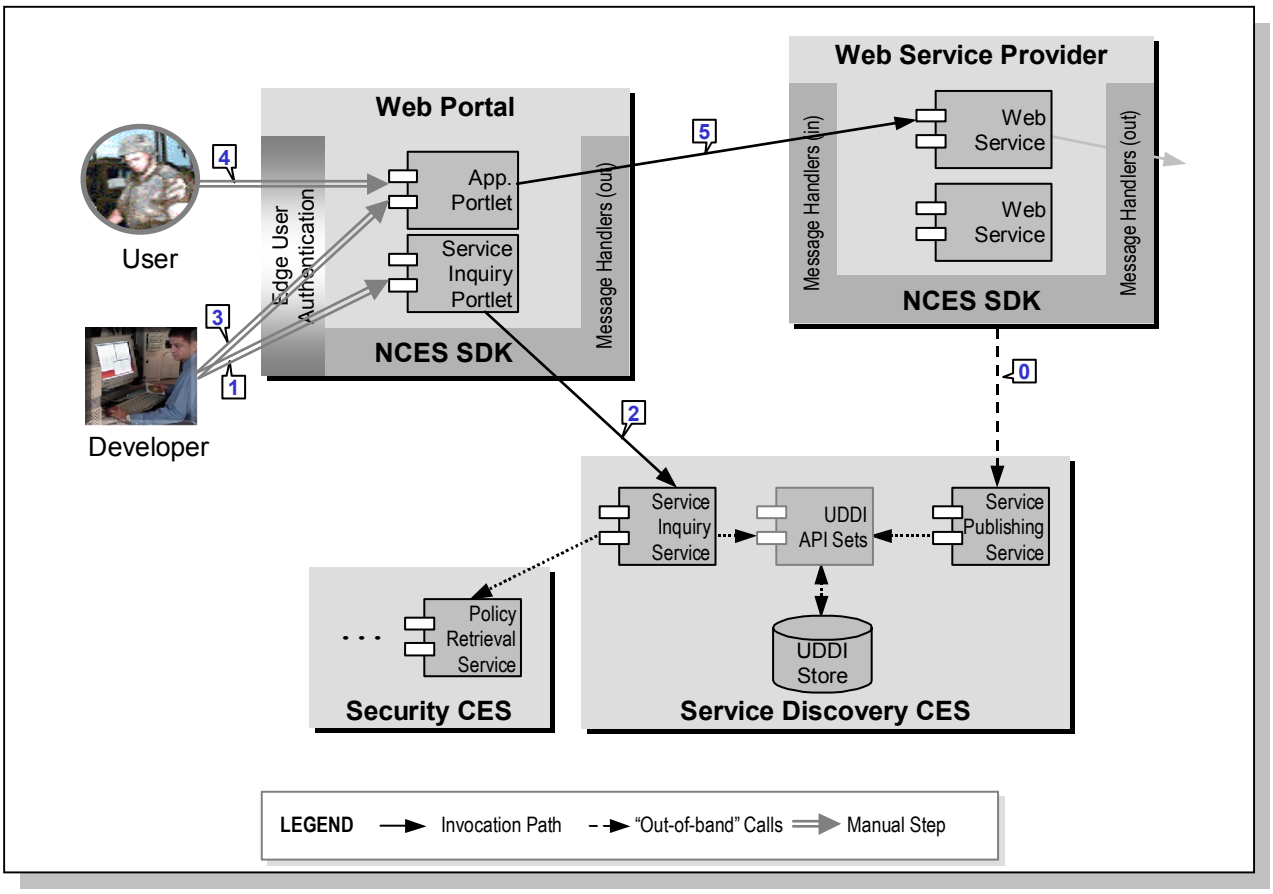


Figure 8: Static Service Discovery Scenario

This scenario involves the following steps:

Step 0. A service provider publishes a Web Service (along with its instance(s)) to the UDDI registry through the Service Publishing Service. This step happens in advance and is therefore dubbed step "0".

Step 1. The developer of an Application Portlet goes to the Service Inquiry Portlet to browse and / or search for the service to be consumed by the application portlet.

Step 2. The Service Inquiry Portlet creates a service inquiry and sends it to the Service Inquiry Service, which returns the matching service instance(s).

Step 3. The developer, at design time, configures the Application Portlet with the retrieved end point information for the target service instance. The configured portlet is then deployed in the portal.

Step 4. After the Application Portlet is up and running, an end user logs on to the portal and starts using the portlet.

Step 5. The portlet, serving as a Web Service consumer, binds to the target service instance and makes a SOAP request on behalf of the user.

Note that, in this scenario, the service inquiry occurs at application design time and the discovered service information is static and "hardwired" into the service consumer.

5.2. Dynamic Service Discovery

In a service-oriented environment, dynamic service discovery can help make system-to-system communications more loosely coupled by eliminating hardwired connections. Figure 9 builds on the previous scenario to illustrate dynamic service discovery.

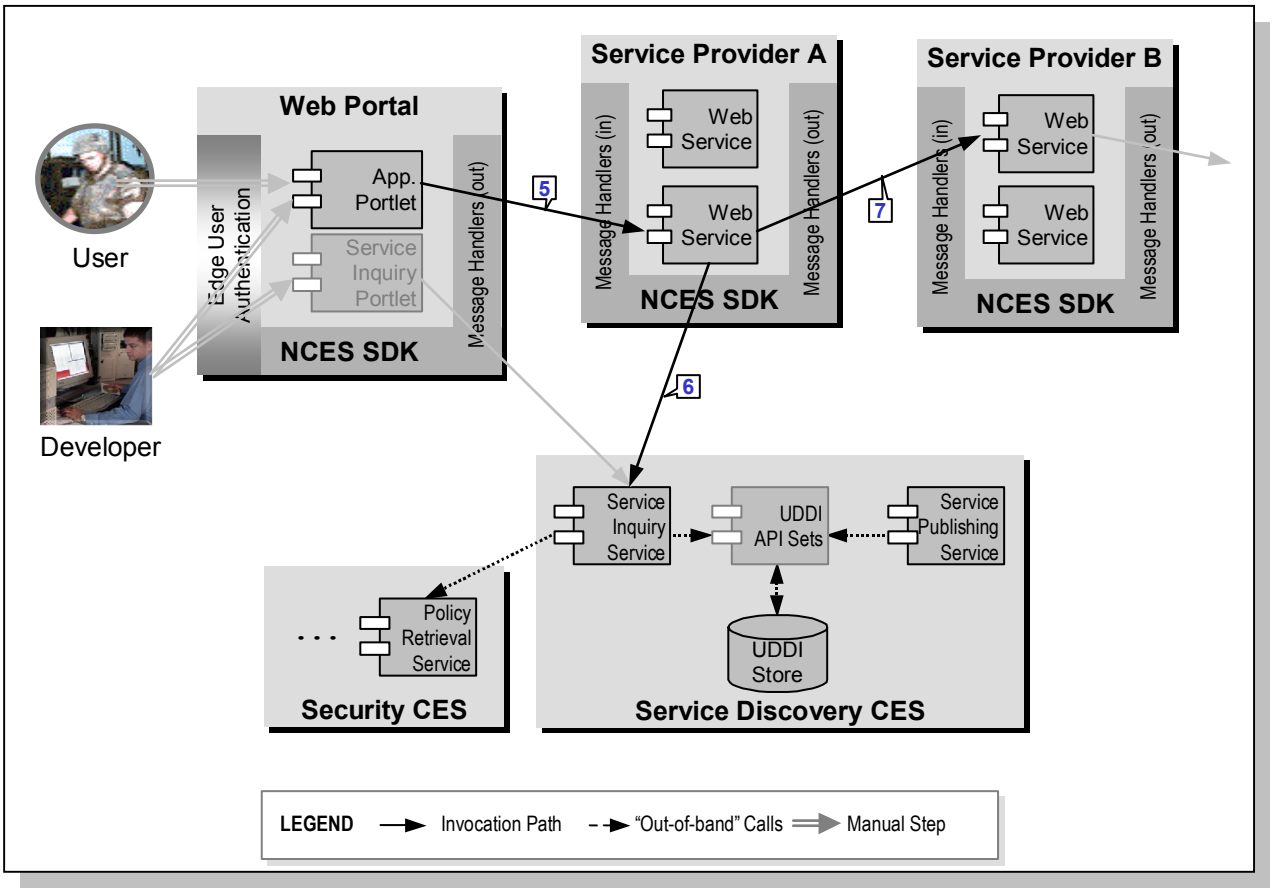


Figure 9: Dynamic Service Discovery Scenario

The diagram depicts the following additional steps:

Step 5. From the previous scenario, the Application Portlet makes a SOAP request on behalf of the user to the service from Service Provider A.

Step 6. The service in Provider A finds it necessary to invoke another Web Service to fulfill the user's request. The service sends a dynamically constructed search request to the Service Inquiry Service, which returns the matching service instance(s) that meet the search criteria.

743 Step 7. The service uses the returned end point information from SIS to invoke another
744 service from Service Provider B.

745 Step 7 may be repeated multiple times if the inquiry returns more than one service instances.

6. SERVICE DISCOVERY INFORMATION MODEL

This section prescribes how discovery information, such as service definitions and metadata, are modeled in the service registry. The goal is to provide a usage "profile" for technology standards that not only suits DoD service discovery needs, but also preserves interoperability across the Net-Centric, heterogeneous environment. The information model defined here is quite minimalistic; it only serves as a starting point for future evolution, driven by the NCES communities. In addition, in this version we focus on the UDDI registry only.

6.1. Basic Information Model

The notion of ***Service Providers***, ***Services***, and ***Service Instances*** are already introduced earlier in the document. Figure 10 depicts the conceptual relationships among those discovery entities as well as the types of metadata with which they are typically associated.

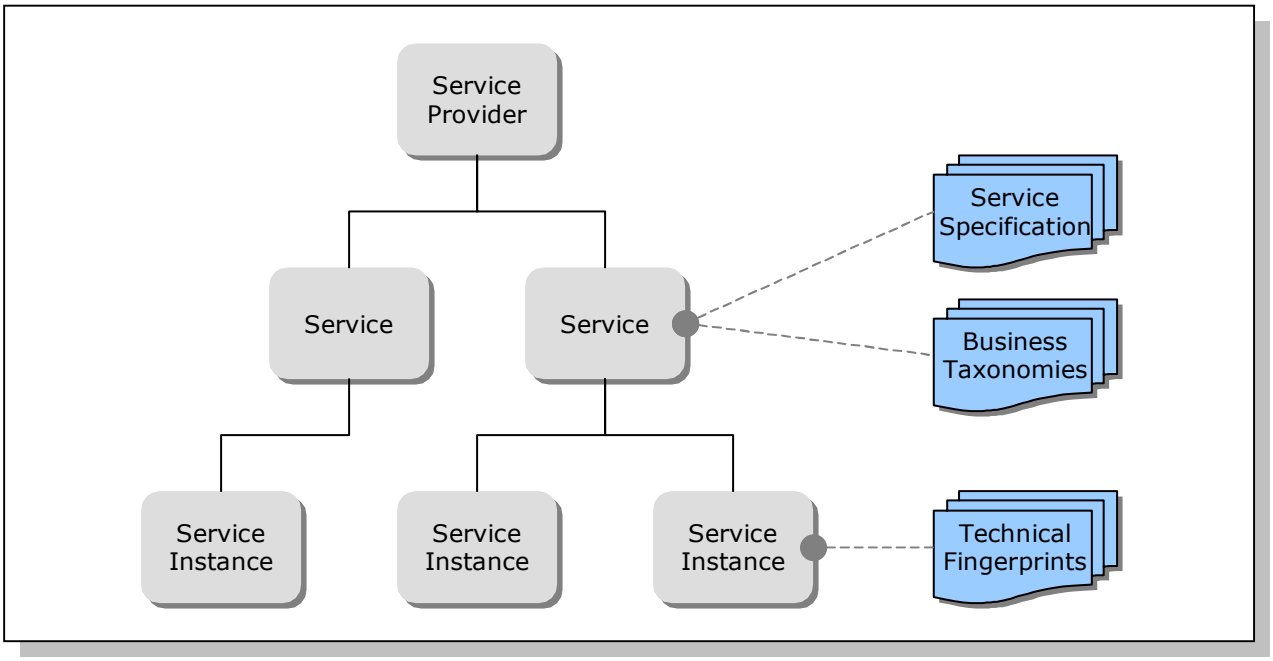


Figure 10: Basic Information Model

For services, in addition to categorize them with business taxonomies, it is a best practice to declare the functional ***Service Specification(s)*** that a service conforms to. A service specification is the interface "contract" for a set of well-defined business operations. For Web Services, a specification is generally written in WSDL format, with the business functionality defined as portTypes. It may also be accompanied by other, more verbose documentation. The specifications are generally implementation-agnostic.

Declaring conformance to specifications has important benefits to service discovery, especially to dynamic discovery at runtime. Take federated search for example. A federated search service distributes content search requests to many data services. Because new data services will be added and existing data sources may go offline, the federated search service can only rely on the

service registry to discover available data sources dynamically. However, not all data sources have the compatible interfaces for the federated search service to consume their data. By requiring the data services declaring their compliance (or non-compliance) to the Federated Search Specification, the search engine can easily discover the available data sources that are accessible.

For service instances, technical "fingerprints" are used to declare the system capabilities and access mechanisms, as outlined in Section 4.2.3. Such technical information is usually specific to a particular deployment of a service, and is thus kept at the instance level.

6.2. Modeling Discovery Entities in UDDI

This subsection defines the mapping rules for representing the discovery information model in UDDI. Unless specified otherwise, the following discussions are based on the UDDI Version 2.03 Data Structure Reference [UDDI-2DS]. The exceptions are some UDDI Version 3.0 features [UDDI-3], which are pointed out explicitly.

6.2.1 Identifiers and Names

In UDDI, the discovery entities are persisted and accessed individually using unique identifiers, or keys. In UDDI 2.0, the keys take the form of a Universally Unique ID (UUID). In addition, the spec mandates that only the UDDI registry can generate keys. Version 3.0 removes this restriction and further allows URI based key formats, but in this document we stick to the 2.0 spec for the time being.

- **RULE_010**: *Discovery entities MUST be identified using UUIDs generated by the UDDI registry.*

The human readable names of the discovery entities should comply with the following rules:

- **RULE_020**: *Human readable names of discovery entities SHOULD expand abbreviations or acronyms to their full word representations.*
- **RULE_030**: *Human readable names of discovery entities SHOULD include a parenthesized list of acronyms or aliases for searching.*

E.g., "Global Directory Service (GDS)".

6.2.2 Service Providers

Service providers are represented as a businessEntity in UDDI. Please see [UDDI-2DS] for more details of the data structure.

- **RULE_110**: *A service provider MUST be represented in a UDDI registry by a businessEntity data structure.*
- **RULE_120**: *A service provider SHOULD provide a businessEntity discovery URL with a use type of "homepage", that points to the provider's HTTP-accessible web home page.*
- **RULE_130**: *A businessEntity description MUST be provided.*

- **RULE_140**: *A businessEntity description SHOULD summarize the main services provided by the service provider.*
- **RULE_150**: *A service provider MUST provide the primary contact information for an individual that can be contacted for service discovery purposes. A secondary contact SHOULD be provided in case the primary contact is not available.*

6.2.3 Services

A logical service is represented by a businessService structure in UDDI. Each businessService is a logical child of a single businessEntity. Please see [UDDI-2DS] for more details of the data structure.

- **RULE_210**: *A service, such as a Web Service, MUST be represented in a UDDI registry by a businessService structure. Only one service can be modeled by an individual businessService.*
- **RULE_220**: *A businessService description MUST be provided. The description SHOULD include the system entities (e.g. portal user), actions (e.g. get, update, etc.), and objects (e.g. track data) that the service performs. The description SHOULD also include the major databases or systems of record that it accesses.*

A logical service is represented by zero or more physical service entry points.

- **RULE_230**: *The businessService MUST include a bindingTemplate element for each service entry point.*

The businessService construct contains a CategoryBag element that allows the service to be categorized according to the available taxonomy based classification schemes. APPENDIX A lists a number of required taxonomies for enterprise services. Additional taxonomies need to be defined later through the community process.

- **RULE_240**: *The businessService MUST include in its CategoryBag element keyed references to all the required enterprise service taxonomies defined in the Appendix A.1 of this document.*

In particular, a service is recommended to declare the service specification(s) it implements. The specifications are defined using the "`nces-mil:discovery:serviceSpecification`" taxonomy (see Appendix A.1).

- **RULE_250**: *The businessService SHOULD declare the service specification(s) it supports using the "`nces-mil:discovery:serviceSpecification`" taxonomy.*

6.2.4 Service Instances

In UDDI, a bindingTemplate construct represents a service instance. The most important information contained in a bindingTemplate is the service entry point, which is either the direct protocol access point for the service or an indirection mechanism that leads to the access point. Please see [UDDI-2DS] for details of the bindingTemplate structure.

840 ➢ **RULE_310**: *A Web Service port MUST be represented by a bindingTemplate in a UDDI*
841 *registry.*

842 The accessPoint element is usually a URL to the service instance's entry point. A companion
843 attribute called URLType is provided to facilitate searching for entry points with a particular
844 protocol. Note that the "protocol" here may not necessarily refer to a network communications
845 protocol. An example might be a Report Ordering service that provides three entry points, one
846 for HTTP, one for email, and one for fax ordering.

847 ➢ **RULE_320**: *A bindingTemplate access point of URLType "http" or "https" MUST*
848 *resolve to the service instance endpoint.*

849 Categorizations may also be applied to bindingTemplates to indicate the technical "fingerprints"
850 of the service instance. This is a UDDI 3.0 feature. In addition to standard UDDI categories
851 such as protocol and transport, the service instance also needs to have the required technical (e.g.
852 security, QoS) categorizations based on the required NCES taxonomies (see Appendix A.2).

853 ➢ **RULE_330**: *The bindingTemplate MUST include in its CategoryBag element keyed*
854 *references to all the required enterprise service taxonomies defined in the Appendix A.2*
855 *of this document.*

856 ## 6.3. Mapping WSDL Descriptions to UDDI

857 For Web Services, WSDL is used to define the logical operations of a service as well as their
858 "bindings" to a particular transport protocol. The OASIS Technical Note "Using WSDL in a
859 UDDI Registry, Version 2.0" [UDDI-WSDL] describes the recommended approach to mapping
860 WSDL descriptions to the UDDI data structures.

861 ➢ **RULE_410**: *A WSDL document represented in UDDI MUST follow the data mapping*
862 *rules defined by the OASIS Technical Note "Using WSDL in a UDDI Registry, Version*
863 *2.0".*

864 Table 2 summarizes the WSDL to UDDI mapping.

865 **Table 2: Summary of WSDL to UDDI Mapping**

WSDL	UDDI
PortType	tModel (categorized as portType)
Abstract WSDL document	OverviewURL of portType tModel
Binding	tModel (categorized as binding and wsdlSpec)
Concrete of WSDL document	OverviewURL of binding tModel
Protocol from binding extension	keyedReference in binding categoryBag
Transport from binding extension (if any)	keyedReference in binding categoryBag
Service	businessService (categorized as service)
Namespace of Service	keyedReference in businessService categoryBag
Port	bindingTemplate

7. INTEGRATION WITH SECURITY SERVICES

7.1. Securing the Service Interfaces

One of the benefits of having the Service Publishing Service and Service Inquiry Service on top of UDDI is to be able to easily leverage the Security CES to provide secure service discovery.

Figure 11 illustrates using SOAP message handlers to secure the Inquiry Service.

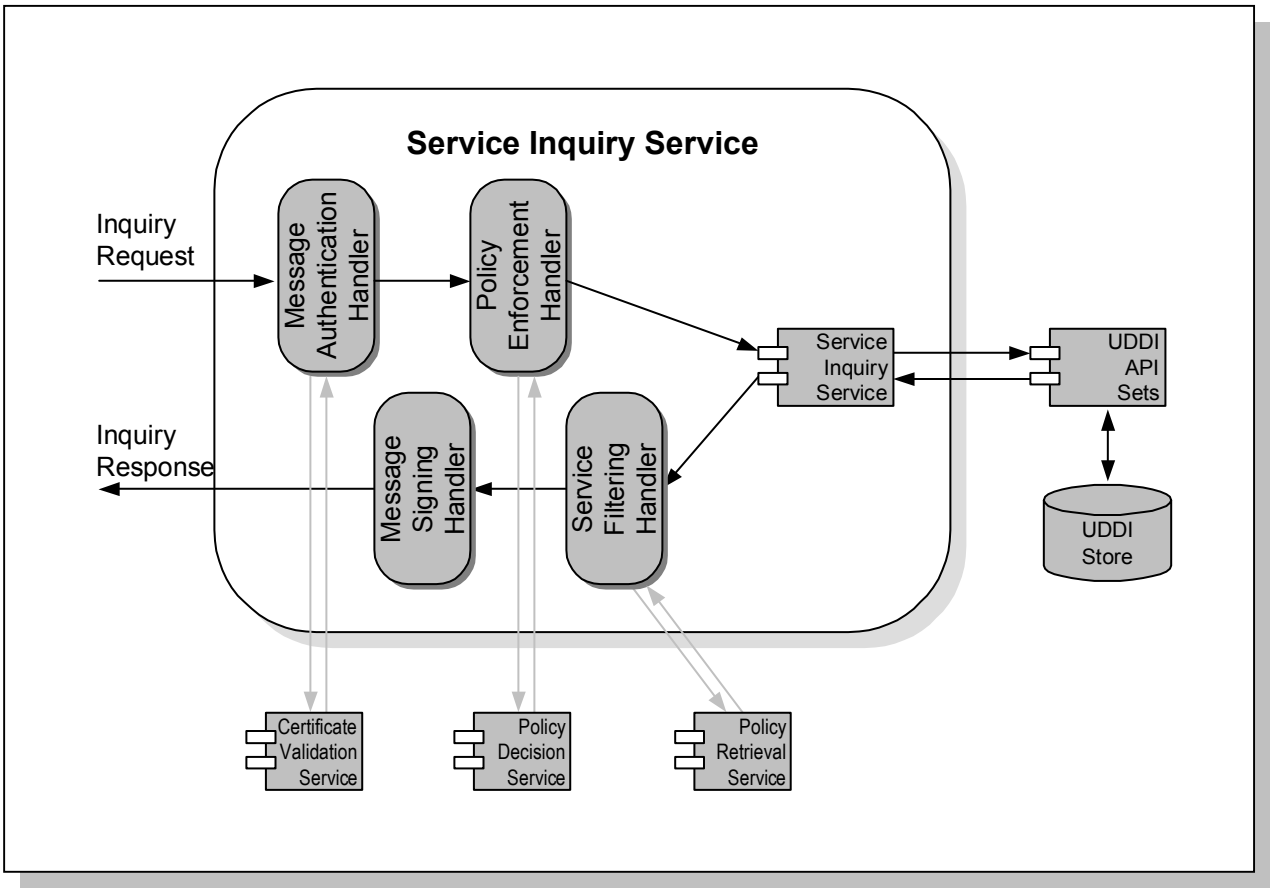


Figure 11: Secure Service Inquiry

The Message Authentication Handler verifies the inquiry request signature, the Policy Enforcement Handler authorizes the inquiry based on the service level authorization policies, and the Message Signing Handler signs the inquiry response – all in the same matter as described in the Security CES architecture. In addition, a Service Filtering Handler is used to provide data level access control: Only the service entries that are visible to the consumer are returned, based on the role based policies.

7.2. Protecting Discovery Entities

Another security aspect is to ensure the integrity and authenticity of the discovery entities published in UDDI. UDDI Version 3.0 specification allows digital signatures to be applied to

just about any UDDI data structure, such as businessService, bindingTemplates, and tModels. Inquirers of a registry can now filter their queries, only requesting data that has in fact been signed. When an inquirer then retrieves and verifies data from a registry, the inquirer can be confident that the data is exactly as the publisher intended it. Similarly, publishers to a registry now have the assurance that they are not being misrepresented by someone claiming to own a UDDI entity. Once publishers have signed data, they can have confidence in the integrity of that data.

Due to the lack of vendor implementations, this feature is currently not supported and will be added in the near future.

7.3. Discovery of Service QoP Requirements

By modeling the Qualify of Protection (QoP) requirements of services using taxonomy-based categorization schemes, these requirements can be just as easily queried as other service metadata. For example, the outbound message handler can check with the service registry to find out whether the target service instance requires message authentication. If not, the message signing step may be skipped to eliminate unnecessary overhead.

The required QoP taxonomies are defined in Appendix A.2.2.

8. FUTURE WORK

The following are identified as items that need to be addressed in future iterations of the architecture:

1. **Federation / Affiliation of Multiple Registries.**

As described in Section 4.5.5, a centralized enterprise service registry will not be adequate to satisfy ubiquitous discovery needs and replication can only scale to a limited extent. Multiple affiliated or "federated" registries, each serving a community but formed by the Service Discovery CES into a "virtual" registry, is a topic that will be addressed in the next version of the document.

2. **Integration with Enterprise Service Management (ESM).**

UDDI registry may also be used to store service performance metrics. Integration with the ESM CES will be addressed in more details later.

3. **Native UDDI Integration of NCES Security Policies.**

As mentioned earlier, UDDI Version 3.0 supports native integration of external security policies, which may bring improved performance compared to the filtering handler approach.

4. **Mapping Security Policies from WSDL to UDDI.**

The OASIS Technical Note on mapping WSDL descriptions to UDDI only covers the basic WSDL elements, not WSDL extensions. For automated service publishing, additional rules need to be defined to map WS-PolicyAttachment or other types of extensions to UDDI data structures.

5. **Support for DDMS.**

The DDMS schema captures many essential resource metadata attributes such as security classification, title, date, key words, formats, and temporal and spatial constraints. By obtaining a DDMS "descriptor" document for a service at publishing time, the registry can automatically create matching tModels and categories for these attributes, which then becomes discoverable information. This may prove to be a much more effective approach for publishing service metadata, compared to individual users manually typing it in via a web interface.

6. **Support for ebXML Registries.**

To be addressed when necessary.

7. **Further Definition of Enterprise Service Taxonomies.**

928 Including both business / functional taxonomies and technical taxonomies. This needs to be
929 driven by the community.

930 **8. Capturing Metadata for Information Consumers.**

931 So far we have discussed publishing / advertising of metadata for information producers, but
932 metadata for information consumers may also need to be captured, such as consumer needs
933 (i.e., "Wanted Ads") and capabilities (e.g. device info, bandwidth, push vs. pull modes, etc.).

APPENDIX A PREDEFINED TAXONOMIES

Taxonomies and identifiers systems play an important role within UDDI. It is through categorization and identification that Service Consumers are able to find services that meet their needs. For a full description of the data structures involved in establishing categorization information, please see [UDDI-2DS].

The taxonomies defined here are currently tentative; they need to be further refined and augmented by the NCES community

A.1 Business / Functional Taxonomies

Name	Description	Applies To	Checked	Current Valid Values
`nces-mil:discovery: serviceSpec`	Identifies the service specification	Service	Yes	TBD
`nces-mil:discovery: serviceSpecType`	Indicates the type of the service specification	Service	Yes	WSDL Schema DTD Text
`nces-mil:discovery: enterpriseServiceType`	Indicates the enterprise service type	Service	Yes	Application Service Mediation CES Messaging CES ESM CES …
`nces-mil:discovery: COI`	Indicates the Community of Interest for the service	Service	Yes	C2 Logistics Intelligence …

A.2 Technical Taxonomies

A.2.1 General

Name	Description	Applies To	Checked	Current Valid Values
`nces-mil:discovery: deploymentEnvironment`	Identifies the deployment environment	Service Instance	Yes	Windows Linux UNIX Other
`nces-mil:discovery: enterpriseRelease`	The release number of the service instance	Service Instance	Yes	R0.3 R0.4 R0.5 ...

A.2.2 Security

Name	Description	Applies To	Checked	Current Valid Values
`nces-mil:security: classification`	Indicates the (system high) security classification	Service Instance	Yes	Secret Unclassified
`nces-mil:security: certificationStatus`	Indicates the C&A status of the deployment	Service Instance	Yes	ATO ATO Pending None
`nces-mil:security: authenticationMethod`	Indicates the required authentication method	Service Instance	Yes	X.509-PKI None

Other technical taxonomies, such as Quality of Service (QoS), will be defined later.

A.3 Defining New Taxonomies

New taxonomies MUST following the best practice outlined in the OASIS UDDI Technical Note, "Providing a Taxonomy for Use in UDDI 2.0" [UDDI-TAX].

APPENDIX B MESSAGE EXAMPLES

B.1 Simple Inquiry Examples – UDDI API vs. Inquiry Service

The following simple examples show how to find a service in a UDDI registry, given the service's WSDL qualified name (i.e., namespace and local service name).

Using UDDI API:

```
<find_service generic="2.0" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference  tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
                     keyName="WSDL type"
                     keyValue="service" />
    <keyedReference  tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
                     keyName="service namespace"
                     keyValue="http://example.com/stockquote/" />
    <keyedReference  tModelKey="uuid:2ec65201-9109-3919-9bec-c9dbefcaccf6"
                     keyName="service local name"
                     keyValue="StockQuoteService" />
  </categoryBag>
</find_service>
```

Using the Inquiry Service provided by Discovery CES:

```
<GetServiceByQName xmlns="urn:nces-mil:discovery:0.4">
  <serviceQName nameSpace="http://example.com/stockquote/">
    StockQuoteService
  </serviceQName>
</GetServiceByQName>
```

The latter is obviously much simpler and user-friendly.

B.2 Simple Inquiry Examples – UDDI API vs. Inquiry Service

A complex service inquiry example is shown below. The inquiry is for an application service that supports the FederatedSearch service specification, can be accessed by SOAP 1.1, and requires X.509 certificate based authentication.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example shows a more complicated inquiry, the result of
the search will be a list of services, each contains
service instances that match the following criteria:
The returned services all have the following attributes:
  1. service type = ApplicationService
     (defined in the taxonomy "nces-mil:discovery:enterpriseServiceType")
  2. service specification = FederatedSearch
     (defined in the taxonomy "nces-mil:discovery:serviceSpec".)
AND the search will further filter on the service instances,
only return service instances that meet the following instance properties:
  1. protocol = SOAP 1.1
     (defined as uddi-org:protocol:soap in property taxonomy
```

```
     "uddi-org:wsdl:categorization:protocol")
   2. authenticationMethod = X.509-PKI
     (defined in the taxonomy "nces-mil:security:authenticationMethod")
-->
<FindServices xmlns="urn:nces-mil:discovery:0.4">
  <serviceLocator>
    <serviceFilter>
      <matchingAttributes>
        <serviceAttribute attributeType="nces-mil:discovery:
enterpriseServiceType">
          <name>service type</name>
          <value>ApplicationService</value>
        </serviceAttribute>
        <serviceAttribute attributeType="nces-mil:discovery:serviceSpec">
          <name>data type</name>
          <value>FederatedSearch</value>
        </serviceAttribute>
      </matchingAttributes>
    </serviceFilter>
    <instanceFilter>
      <matchingProperties>
        <instanceProperty propertyType="uddi-
org:wsdl:categorization:protocol">
          <name>protocol</name>
          <value>uddi-org:protocol:soap</value>
        </instanceProperty>
        <instanceProperty propertyType="nces-
mil:security:authenticationMethod">
          <name>authenticationMethod</name>
          <value>X.509-PKI</value>
        </instanceProperty>
      </matchingProperties>
    </instanceFilter>
  </serviceLocator>
</FindServices>
```

B.3 Service Publishing

This is an example for publishing a service and its instance using the Publishing Service. Please note the inclusion of required functional and technical categorizations.

```
<?xml version="1.0" encoding="UTF-8"?>
<PublishService xmlns="urn:nces-mil:discovery:0.4">
  <serviceToPublish publicVisibility="true">
    <name>MySearchService</name>
    <qualifiedName nameSpace="http://some.agency.mil/search">
      MySearchService
    </qualifiedName>
    <description>This is a sample SOAP service to show how a service can
be published using the Publishing Service.</description>
    <identifier/>
    <attributes>
      <serviceAttribute attributeType="nces-mil:discovery:
enterpriseServiceType">
        <value>ApplicationService</value>
      </serviceAttribute>
```

```
        <serviceAttribute attributeType="nces-mil:discovery:serviceSpec">
          <value>FederatedSearch</value>
        </serviceAttribute>
      </attributes>
      <instances>
        <serviceInstance>
          <identifier/>
          <serviceIdentifier/>
            <accessPoint>
              http://merceds:7001/search/services/MySearchService
            </accessPoint>
          <properties>
            <instanceProperty>
              <name>deployer name</name>
              <value>BAH lab</value>
            </instanceProperty>
            <instanceProperty>
              <name>deployer contact</name>
              <value>1-800-JohnDoe</value>
            </instanceProperty>
            <instanceProperty propertyType="uddi-
org:wsdl:categorization:transport">
              <name>transport</name>
              <value>uddi-org:protocol:http</value>
            </instanceProperty>
            <instanceProperty propertyType="uddi-
org:wsdl:categorization:protocol">
              <name>protocol</name>
              <value>uddi-org:protocol:soap</value>
            </instanceProperty>
            <instanceProperty propertyType="nces-
mil:security:authenticationMethod">
              <name>authenticationMethod</name>
              <value>X.509-PKI</value>
            </instanceProperty>
            <instanceProperty propertyType="nces-
mil:esm:QoS:minBandWidthSupported">
              <name>minBandWidthSupported</name>
              <value>T1</value>
            </instanceProperty>
          </properties>
        </serviceInstance>
      </instances>
    </serviceToPublish>
</PublishService>
```

This page was intentionally left blank.

APPENDIX C REFERENCES

[JAXR]	Java API for XML Registries (JAXR), http://java.sun.com/xml/jaxr/index.jsp
[JAXRPC]	Java API for XML-Based RPC (JAX-RPC) Specification 1.0 http://java.sun.com/xml/jaxrpc/docs.html
[RFC2119]	S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt
[RFC2828]	RFC 2828, Internet Security Glossary http://www.ietf.org/rfc/rfc2828.txt
[SAML]	Security Assertion Markup Language http://www.oasis-open.org/committees/security/#documents
[SECARCH]	NCES Security Architecture document, v0.3, prepared by Booz Allen Hamilton for DISA, March 3, 2004
[SOAP]	Simple Object Access Protocol 1.1 http://www.w3.org/TR/SOAP/
[UDDI-2DS]	UDDI Version 2.03 Data Structure Reference, http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm
[UDDI-3]	UDDI Version 3.0 Specification, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm
[UDDI-TAX]	"Providing a Taxonomy for Use in UDDI Version 2", http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-taxonomy-provider-v100-20010717.htm
[UDDI-WSDL]	"Using WSDL in a UDDI Registry, Version 2.0", UDDI Technical Note, November 3, 2003, http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm
[WS-DISC]	Web Services Dynamic Discovery (WS-Discovery), Feb. 2004, http://msdn.microsoft.com/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnglobspec/html/ws-discovery.asp
[WS-GLOS]	Web Service Glossary http://www.w3.org/TR/ws-gloss/
[WS-I]	WS-Interoperability Initiative http://www.ws-i.org/
[WSS]	Web Services Security: SOAP Message Security Spec 1.0 (Community Draft), http://www.oasis-open.org/apps/org/workgroup/wss/
[XACML]	XML Access Control Markup Language (XACML), Version 1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[XKMS]	XML Key Management Specification http://www.w3.org/TR/xkms
[XMLDSIG]	XML Signatures Syntax and Processing http://www.w3.org/TR/xmldsig-core/

[XMLENC]	XML Encryption Syntax and Processing http://www.w3.org/TR/xmlenc-core/

1092